Credit Scoring Dataset

The dataset being used describes the customers (seniority, age, marital status, income, and other characteristics), the loan (the requested amount, the price of the idem) as well as its status (was it paid back or not).

The Dataset is available on GitHub at [https://github.com/gastonstat/CreditScoring/ (https://github.com/gastonstat/CreditScoring/)](https://github.com/gastonstat/CreditScoring/)

This is a binary classification problem, and three kinds of algorithms/tools will be utilized:

1. DecisionTreeClassifier (from scikit-learn)
2. RandomForestClassifier (from scikit-learn)
3. Extreme Gradient Boosting (from xgboost library)

In [78]:

```
import os, sys, io, requests
import numpy as np
import pandas as pd
import sklearn
import xgboost as xgb
import matplotlib
```

In [79]:

```
#Download the Dataset and store in /data/raw
url = 'https://github.com/gastonstat/CreditScoring/raw/master/CreditScoring.csv'
save_file_path = os.path.join('data', 'raw', 'CreditScoring.csv')

#Download the file if it does not already exist
if not os.path.exists(save_file_path):
    file_stream = requests.get(url, allow_redirects=True, stream=True)
    with open(save_file_path, 'wb+') as save_file:
        #save_file.write(file_stream)
        for chunk in file_stream.iter_content(chunk_size=1024 * 8):
                    if chunk:
                        save_file.write(chunk)
                        save_file.flush()
                        os.fsync(save_file.fileno())
```

In [80]:
```
#import the csv file into a pandas dataframe
column_names = ["Status","Seniority","Home","Time","Age","Marital","Records","Jot
                "Assets","Debt","Amount","Price"]

df = pd.read_csv(save_file_path, names=None)
df.head()
```

Out[80]:

	Status	Seniority	Home	Time	Age	Marital	Records	Job	Expenses	Income	Assets	Debt
0	1	9	1	60	30	2	1	3	73	129	0	0
1	1	17	1	60	58	3	1	1	48	131	0	0
2	2	10	2	36	46	2	2	3	90	200	3000	0
3	1	0	1	60	24	1	1	1	63	182	2500	0
4	1	0	1	36	26	1	1	1	46	107	0	0

In [81]:
```
df.describe()
```

Out[81]:

	Status	Seniority	Home	Time	Age	Marital	Records
count	4455.000000	4455.000000	4455.000000	4455.000000	4455.000000	4455.000000	4455.000000
mean	1.281257	7.987205	2.657015	46.441751	37.077666	1.879012	1.173513
std	0.450162	8.173444	1.610467	14.655225	10.984856	0.643748	0.378733
min	0.000000	0.000000	0.000000	6.000000	18.000000	0.000000	1.000000
25%	1.000000	2.000000	2.000000	36.000000	28.000000	2.000000	1.000000
50%	1.000000	5.000000	2.000000	48.000000	36.000000	2.000000	1.000000
75%	2.000000	12.000000	4.000000	60.000000	45.000000	2.000000	1.000000
max	2.000000	48.000000	6.000000	72.000000	68.000000	5.000000	2.000000

In [82]:
```
#Check how many nulls there are, and decide a manner of filling these nulls in.
df.isna().sum()
```

Out[82]:
```
Status        0
Seniority     0
Home          0
Time          0
Age           0
Marital       0
Records       0
Job           0
Expenses      0
Income        0
Assets        0
Debt          0
Amount        0
Price         0
dtype: int64
```

In [83]:
```
#reduce the column names to lower case for convention's sake
df.columns = df.columns.str.lower()
df.head()
```

Out[83]:

	status	seniority	home	time	age	marital	records	job	expenses	income	assets	debt	am
0	1	9	1	60	30	2	1	3	73	129	0	0	
1	1	17	1	60	58	3	1	1	48	131	0	0	
2	2	10	2	36	46	2	2	3	90	200	3000	0	
3	1	0	1	60	24	1	1	1	63	182	2500	0	
4	1	0	1	36	26	1	1	1	46	107	0	0	

The column significance, according to the dataset source:

1 Status --- credit status 2 Seniority --- job seniority (years) 3 Home --- type of home ownership 4 Time --- time of requested loan 5 Age --- client's age 6 Marital --- marital status 7 Records --- existance of records 8 Job --- type of job 9 Expenses --- amount of expenses 10 Income --- amount of income 11 Assets --- amount of assets 12 Debt --- amount of debt 13 Amount --- amount requested of loan 14 Price --- price of good

Status is a categorical field that has been encoded as follows: “1” means “OK”, and the value “2” means “default”, and “0” means that the value is missing.

In [84]:
```
#make dictionaries that will map from the encoding to their categorical values tc

#for column status
status_column_mapping = {
    0 : 'unknown',
    1 : 'ok',
    2 : 'default'
}

#for column home
home_column_mapping = {
    1: 'rent',
    2: 'owner',
    3: 'private',
    4: 'ignore',
    5: 'parents',
    6: 'other',
    0: 'unknown'
}

#for column marital
marital_column_mapping = {
    1: 'single',
    2: 'married',
    3: 'widow',
    4: 'separated',
    5: 'divorced',
    0: 'unknown'
}

#for column records
records_column_mapping = {
     1: 'no',
     2: 'yes',
     0: 'unknown'
}

#for column job
job_column_mapping = {
    1: 'fixed',
    2: 'partime',
    3: 'freelance',
    4: 'others',
    0: 'unknown'
}

def dictForStr(dict_name: str) :
    if dict_name=='home':
        return home_column_mapping
    elif dict_name=='job':
        return job_column_mapping
    elif dict_name=='status':
        return status_column_mapping
    elif dict_name=='marital':
        return marital_column_mapping
    elif dict_name=='records':
```

```python
        return records_column_mapping

df_full = df.copy()
cols = ['home', 'job', 'status', 'marital', 'records']
for c in cols:
    df_full[c] = df_full[c].map(dictForStr(c))



df_full.head()
```

Out[84]:

	status	seniority	home	time	age	marital	records	job	expenses	income	assets	debt
0	ok	9	rent	60	30	married	no	freelance	73	129	0	0
1	ok	17	rent	60	58	widow	no	fixed	48	131	0	0
2	default	10	owner	36	46	married	yes	freelance	90	200	3000	0
3	ok	0	rent	60	24	single	no	fixed	63	182	2500	0
4	ok	0	rent	36	26	single	no	fixed	46	107	0	0

In [85]:
```
for col in ['home', 'job', 'status', 'marital', 'records'] :
    print("value_count for column", col, "\n",df_full[col].value_counts(), "\n---
```

```
value_count for column home
 owner      2107
rent        973
parents     783
other       319
private     247
ignore       20
unknown       6
Name: home, dtype: int64
---------

value_count for column job
 fixed        2806
freelance    1024
partime       452
others        171
unknown         2
Name: job, dtype: int64
---------

value_count for column status
 ok         3200
default    1254
unknown       1
Name: status, dtype: int64
---------

value_count for column marital
 married      3241
single        978
separated     130
widow          67
divorced       38
unknown         1
Name: marital, dtype: int64
---------

value_count for column records
 no     3682
yes     773
Name: records, dtype: int64
---------
```

In [86]:
```python
#status is the label, hence unknown values are removed
df_full = df_full[df_full.status != 'unknown']

#ensure the label col has usable values
df_full['status'].value_counts()
```

Out[86]:
```
ok         3200
default    1254
Name: status, dtype: int64
```

In [87]:
```python
#columns income, assets and debt are numeric columns that have the value 99999999
#convert these to np.nan for a clearer understanding
for c in ['income', 'assets', 'debt']:
    df_full[c] = df_full[c].replace(to_replace=99999999, value=np.nan)
```

In [88]:
```python
#View the distribution of the numerical columns of the full, un-encoded dataset
df_full.describe().round()
```

Out[88]:

	seniority	time	age	expenses	income	assets	debt	amount	price
count	4454.0	4454.0	4454.0	4454.0	4420.0	4407.0	4436.0	4454.0	4454.0
mean	8.0	46.0	37.0	56.0	131.0	5404.0	343.0	1039.0	1463.0
std	8.0	15.0	11.0	20.0	86.0	11574.0	1246.0	475.0	628.0
min	0.0	6.0	18.0	35.0	0.0	0.0	0.0	100.0	105.0
25%	2.0	36.0	28.0	35.0	80.0	0.0	0.0	700.0	1117.0
50%	5.0	48.0	36.0	51.0	120.0	3000.0	0.0	1000.0	1400.0
75%	12.0	60.0	45.0	72.0	165.0	6000.0	0.0	1300.0	1692.0
max	48.0	72.0	68.0	180.0	959.0	300000.0	30000.0	5000.0	11140.0

At this point, since ther dataset is well-understood and labels are ok, splitting the dataset between test and training sets will be done. After which cleaning and additional data-prep for training will follow...

Split Srategy:: Training -> 60%, Validation -> 20%, testing -> 20%

In [89]:
```
#use train_test_split from scikit-learn to split the data. First split between 60
#and then the 40% block will be split in two 20% blocks.

from sklearn.model_selection import train_test_split

df_train_, df_test      = train_test_split(df_full, train_size=0.8, test_size=0.2
df_train, df_validation = train_test_split(df_train_, train_size=0.75, test_size=

print(f"The num of records in test set is {len(df_test)}, while the training set 
and the validation set has {len(df_validation)} records")
```

```
The num of records in test set is 891, while the training set has 2672 records
and the validation set has 891 records
```

status == 'default' when the loan was defaulted on. The model is meant to prodict loan defaults. Therefore the label column will have a numeric value of 1 when there is a default (i.e., status == 1, when status == 'default' and status == 0 otherwise).

In [90]:

```python
y_train_ = df_train[df_train['status']=='default']
y_val_   = df_validation[df_validation['status']=='default']

y_train = (df_train.status == 'default').values
y_val = (df_validation.status == 'default').values

y_train
print(y_val)
```

```
[False False  True  True False False False  True  True False  True  True
 False False False False False False  True  True False False False  True
  True False False  True False False False False False False False False
 False False False False False False  True False False  True False False
 False False False False False False  True  True  True False False  True
  True  True False False False False False False  True False False False
 False False False  True False  True False  True  True False  True  True
 False False False False False False  True False False False  True False
 False  True False False False  True  True  True False  True  True False
 False False  True False  True False  True False  True  True  True  True
 False False False False False False False False  True False False False
  True False False False False  True  True  True False False  True False
  True  True False False False False False False  True False  True  True
 False  True False False  True False False  True False  True False False
 False False False False False False False False False False False False
 False False False False  True False  True False  True False  True False
 False False  True  True False False False False  True  True False False
  True  True False False False False False False False False  True False
 False False False False  True False  True  True False False False False
 False False False False  True  True False  True False  True  True False
 False False False False  True  True  True False False False False False
 False  True False False False False False False False False False False
  True  True False  True False  True False  True  True  True False False
 False False False False False False False False False False  True False
  True  True False  True False  True False  True False  True False  True
 False False False False False False  True False False False False False
 False False False False False  True False False  True False False False
  True False False False False  True  True  True  True False False False
 False False False False False False  True False False  True False False
 False  True False  True  True False False False False False False False
  True  True  True  True False  True False  True  True False False False
 False False  True False False  True False False  True  True False False
  True False False False False  True False  True False False False False
 False False False False False  True False False False False  True False
 False  True  True False False  True False False False False  True False
 False False False False  True False  True  True  True False False False
  True False  True  True False  True False False False False  True False
 False  True False False  True False  True  True  True False False False
 False False  True False False False False False False False False  True
  True  True False  True False False False False  True  True  True False
  True False False  True False False False False False False False False
  True False False False False False False  True  True False False  True
 False False False False False False False False False False False  True
  True  True False False False False  True  True False  True False False
 False  True False False False False  True False False  True False  True
 False False False False False False False  True  True  True False False
 False False False False False  True False False False  True False False
```

```
 False False False  True False False False False False  True False  True
  True  True False  True  True  True  True False False False False False
  True False  True False False  True False False False  True  True False
 False  True  True  True False False False False  True  True  True False
  True  True  True  True False False  True False False False  True False
 False  True  True False False False False False False  True False False
 False  True False False False False  True False False  True False False
 False False False False False False False  True False False False False
  True False False  True  True  True  True False  True False False  True
 False False False False False False False False False False False False
 False False False False False False False False False  True  True False
 False False  True False False  True  True  True  True  True False  True
  True False False  True False False  True False False False False False
 False False False  True  True False  True False False  True False False
  True False False False  True False  True False False  True False False
  True False False False False False False False False False False  True
 False  True False  True False  True False False False False False  True
 False False False False False  True  True  True False  True False False
 False False False False  True  True False False False False False False
 False False False False False  True  True False False  True False False
 False  True False False  True False False False False False  True False
 False False  True  True False False False False False False False False
 False False  True  True False False False False False False  True False
  True  True False  True False  True False False False False False False
 False False False False False False False False  True False  True False
 False False False  True False False False False False False False False
 False False False False  True False False False False False False False
 False  True False]
```

In [91]:
```python
#drop the status column to ensure the model does not end up training on this colu
del df_train['status']
del df_validation['status']

df_train.head()
```

Out[91]:

	seniority	home	time	age	marital	records	job	expenses	income	assets	de
2778	10	parents	36	37	single	yes	fixed	35	150.0	0.0	0
3381	7	other	60	27	single	no	fixed	35	78.0	0.0	0
46	1	owner	60	49	widow	no	freelance	35	233.0	15900.0	4500
1008	3	owner	24	29	married	no	fixed	45	254.0	6000.0	800
504	5	other	24	41	separated	no	freelance	35	0.0	0.0	0

In [92]:
```
#deal with missing values: fill all NaNs with 0
#the alternative would be to plug in various kinds of cental-placement figures li
#I'm choosing to make this 0 instead of placing computed values that don't actual

df_train = df_train.fillna(0)
df_validation = df_validation.fillna(0)

#check how many na-s we have among the numeric columns
num_of_na = 0
for col in ['seniority', 'time', 'expenses', 'income', 'assets', 'debt', 'amount'
    num_of_na += df_train[col].isna().sum()

print(f"The total number of NaN in the numeric columns is {num_of_na}")
```

```
The total number of NaN in the numeric columns is 0
```

In [93]:
```
#turn each row into a dictionary
dict_train = df_train.to_dict(orient='records')
dict_val = df_validation.to_dict(orient='records')

#print out one row/dict to see structure
print(dict_train[0])
```

```
{'seniority': 10, 'home': 'parents', 'time': 36, 'age': 37, 'marital': 'singl
e', 'records': 'yes', 'job': 'fixed', 'expenses': 35, 'income': 150.0, 'asset
s': 0.0, 'debt': 0.0, 'amount': 1000, 'price': 2484}
```

In [94]:
```
#The above created dictionaries can be fed into sklearn's DictVectorizer
#from sklear documentation:
#  This transformer turns lists of mappings (dict-like objects) of feature names
#  scipy.sparse matrices for use with scikit-learn estimators.
#  When feature values are strings, this transformer will do a binary one-hot (ak
#  is constructed for each of the possible string values that the feature can tak
#  Note that this transformer will only do a binary one-hot encoding when feature
#  features are represented as numeric values such as int or iterables of strings
#  OneHotEncoder to complete binary one-hot encoding.

from sklearn.feature_extraction import DictVectorizer

#create DictVectorizer object
dv = DictVectorizer(sparse=False) #setting sparce to true would produce scipy.spa

#make feature
X_Train = dv.fit_transform(dict_train)
X_val   = dv.transform(dict_val)

print(X_val)
```

```
[[3.1e+01 6.0e+02 4.5e+03 ... 1.0e+00 1.2e+01 1.8e+01]
 [3.0e+01 5.0e+02 7.0e+03 ... 1.0e+00 7.0e+00 3.0e+01]
 [1.9e+01 1.3e+03 9.5e+03 ... 0.0e+00 1.0e+00 6.0e+01]
 ...
 [6.3e+01 7.0e+02 3.5e+03 ... 0.0e+00 3.7e+01 4.8e+01]
 [3.2e+01 1.4e+03 5.5e+03 ... 0.0e+00 2.0e+00 6.0e+01]
 [4.3e+01 1.1e+03 3.0e+03 ... 1.0e+00 2.0e+00 6.0e+01]]
```

#Decision Tree Section

In [95]:
```
#Since this is a classification task the exact Decision Tree algo being used is t
from sklearn.tree import DecisionTreeClassifier

#create the DecisionTreeClassifier objerct and fit the data
dt = DecisionTreeClassifier()
dt.fit(X_Train, y_train)
```

Out[95]:
```
DecisionTreeClassifier()
```

In [96]:
```
#Now that the model has been trained, it's performance is to be evaluated
#AUC (area under the ROC Curve) is being used as it's a well-regarded tool for bi
from sklearn.metrics import roc_auc_score

#obtain scores to evaluate with auc
y_pred = dt.predict_proba(X_Train)[:,1]
pred_on_training = roc_auc_score(y_train, y_pred)

y_pred = dt.predict_proba(X_val)[:, 1]
pred_on_validation = roc_auc_score(y_val, y_pred)

print(f"The score against training set is {pred_on_training*100:.4f}%,",
      f"while the score against validation set is {pred_on_validation*100:.4f}%.'
```

```
The score against training set is 100.0000%, while the score against validation
set is 64.6655%.
```

Observation: The great efficiency of the model on the training set, as opposed to the validation set, seems to show overfitting. Thus there seems to be a relative lack of the ability to generalize to unkown data.

In [97]:
```
#tweak the max_depth parameter of the DecisiosnTreeClassifier to reduce over-fitt
#max_depth controls the complexity of the tree by putting a limit to the number c
#lesser complexity is hoped to prevent over-fitting of the model to the training

dt = DecisionTreeClassifier(max_depth=2)
dt.fit(X_Train, y_train)
```

Out[97]:
```
DecisionTreeClassifier(max_depth=2)
```

In [98]:
```
#Since the tree itself is being controlled, it makes sense to see the tree genera

from sklearn.tree import export_text

tree_as_text = export_text(dt, feature_names=dv.feature_names_)
print(tree_as_text)
```

```
|--- seniority <= 2.50
|   |--- records=no <= 0.50
|   |   |--- class: True
|   |--- records=no >  0.50
|   |   |--- class: False
|--- seniority >  2.50
|   |--- records=no <= 0.50
|   |   |--- class: False
|   |--- records=no >  0.50
|   |   |--- class: False
```

In [99]:
```python
#obtain scores to evaluate with auc
y_pred = dt.predict_proba(X_Train)[:,1]
pred_on_training = roc_auc_score(y_train, y_pred)

y_pred = dt.predict_proba(X_val)[:, 1]
pred_on_validation = roc_auc_score(y_val, y_pred)

print(f"The score against training set is {pred_on_training*100:.4f}%,",
      f"while the score against validation set is {pred_on_validation*100:.4f}%.'
```

```
The score against training set is 71.4412%, while the score against validation
set is 72.1128%.
```

The lessened score againsdt the training set and the increased score against the validation set indicates lessening of overfitting due to the max_depth parameter being set to 2.

In [100]:
```python
#To continue the process, the model will be tuned for parameter according to two

print('Tune Max_Depth')
#tune max_depth
for depth in list(range(1, 10, 2)):
    dt = DecisionTreeClassifier(max_depth=depth)
    dt.fit(X_Train, y_train)
    y_pred = dt.predict_proba(X_val)[:, 1]
    auc = roc_auc_score(y_val, y_pred)
    print(f"for depth of {depth}, auc is {auc*100:.4f}%")

print('\n\nTune Min_Leaf')
#tune for min_leaf_size
for depth in list(range(1, 10, 2)):
    for min_leaf_size in list(range(1, 10, 2))+[20, 30, 40, 50]:
        dt = DecisionTreeClassifier(max_depth=depth, min_samples_leaf=min_leaf_si
        dt.fit(X_Train, y_train)
        y_pred = dt.predict_proba(X_val)[:, 1]
        auc = roc_auc_score(y_val, y_pred)
        print(f"for depth of {depth} and min_samples_leaf of {min_leaf_size}, the
```

```
Tune Max_Depth
for depth of 1, auc is 64.3967%
for depth of 3, auc is 74.6159%
for depth of 5, auc is 77.5006%
for depth of 7, auc is 71.6741%
for depth of 9, auc is 66.6904%


Tune Min_Leaf
for depth of 1 and min_samples_leaf of 1, the auc is 64.3967%
for depth of 1 and min_samples_leaf of 3, the auc is 64.3967%
for depth of 1 and min_samples_leaf of 5, the auc is 64.3967%
for depth of 1 and min_samples_leaf of 7, the auc is 64.3967%
for depth of 1 and min_samples_leaf of 9, the auc is 64.3967%
for depth of 1 and min_samples_leaf of 20, the auc is 64.3967%
for depth of 1 and min_samples_leaf of 30, the auc is 64.3967%
for depth of 1 and min_samples_leaf of 40, the auc is 64.3967%
for depth of 1 and min_samples_leaf of 50, the auc is 64.3967%
for depth of 3 and min_samples_leaf of 1, the auc is 74.6159%
for depth of 3 and min_samples_leaf of 3, the auc is 74.6159%
for depth of 3 and min_samples_leaf of 5, the auc is 74.6159%
for depth of 3 and min_samples_leaf of 7, the auc is 74.6159%
for depth of 3 and min_samples_leaf of 9, the auc is 74.6159%
for depth of 3 and min_samples_leaf of 20, the auc is 74.6159%
for depth of 3 and min_samples_leaf of 30, the auc is 74.6159%
for depth of 3 and min_samples_leaf of 40, the auc is 74.6159%
for depth of 3 and min_samples_leaf of 50, the auc is 74.6159%
for depth of 5 and min_samples_leaf of 1, the auc is 77.4406%
for depth of 5 and min_samples_leaf of 3, the auc is 76.8247%
for depth of 5 and min_samples_leaf of 5, the auc is 77.0185%
for depth of 5 and min_samples_leaf of 7, the auc is 77.0003%
for depth of 5 and min_samples_leaf of 9, the auc is 77.0003%
for depth of 5 and min_samples_leaf of 20, the auc is 77.5650%
```

```
for depth of 5 and min_samples_leaf of 30, the auc is 76.9824%
for depth of 5 and min_samples_leaf of 40, the auc is 78.2054%
for depth of 5 and min_samples_leaf of 50, the auc is 78.6493%
for depth of 7 and min_samples_leaf of 1, the auc is 72.0631%
for depth of 7 and min_samples_leaf of 3, the auc is 74.7873%
for depth of 7 and min_samples_leaf of 5, the auc is 76.7503%
for depth of 7 and min_samples_leaf of 7, the auc is 76.6881%
for depth of 7 and min_samples_leaf of 9, the auc is 76.9175%
for depth of 7 and min_samples_leaf of 20, the auc is 78.6645%
for depth of 7 and min_samples_leaf of 30, the auc is 78.0176%
for depth of 7 and min_samples_leaf of 40, the auc is 79.2406%
for depth of 7 and min_samples_leaf of 50, the auc is 80.1572%
for depth of 9 and min_samples_leaf of 1, the auc is 67.4410%
for depth of 9 and min_samples_leaf of 3, the auc is 70.6180%
for depth of 9 and min_samples_leaf of 5, the auc is 74.3395%
for depth of 9 and min_samples_leaf of 7, the auc is 75.8207%
for depth of 9 and min_samples_leaf of 9, the auc is 75.9763%
for depth of 9 and min_samples_leaf of 20, the auc is 78.0437%
for depth of 9 and min_samples_leaf of 30, the auc is 78.5501%
for depth of 9 and min_samples_leaf of 40, the auc is 79.2610%
for depth of 9 and min_samples_leaf of 50, the auc is 80.5574%
```

In [101]:

```
#The best auc is given by the following combination
#for depth of 9 and min_samples_leaf of 20, the auc is 80.4157%

#train final model with those parameters
dt = DecisionTreeClassifier(max_depth=9, min_samples_leaf=20)
dt.fit(X_Train, y_train)
```

Out[101]:

```
DecisionTreeClassifier(max_depth=9, min_samples_leaf=20)
```

Random Forest Section follows...

Using the concept of Ensamble Learning, a number of models will be used to derive the verdict. The majority decision of all the verdicts will be taken to be the decision of the ensamble system as a whole.

Ramdom Forests, in particular, work by implementing a number of trees each working on a unique combination of features to reach their classification decision. The majority of these then is taken to be the verdict outputted by the system as a whole. One startegy of selecting features is simply to randomly choose a set of features per tree in the forest.

In [102]:

```
from sklearn.ensemble import RandomForestClassifier

#create a RandomForest object with 10 trees
#the parameter n_estimators sets the number of trees in the random-forest
rf = RandomForestClassifier(n_estimators=10, random_state=42)
rf.fit(X_Train, y_train)
```

Out[102]:

```
RandomForestClassifier(n_estimators=10, random_state=42)
```

In [103]:

```
#Check the performance of the forest
y_pred = rf.predict_proba(X_val)[:, 1]
print(f"The performance of the random forest against the validatio set is {roc_au
```

```
The performance of the random forest against the validatio set is 78.0516%.
```

In [104]:

```
#tune the n_estimator parameter
plot_dict = {}
for n in range(10, 201, 20):
    rf = RandomForestClassifier(n_estimators=n, random_state=42)
    rf.fit(X_Train, y_train)
    y_pred = rf.predict_proba(X_val)[:, 1]
    s = roc_auc_score(y_val, y_pred)*100
    plot_dict[n] = s
    print(f"for n_estimator={n}, auc is {s:.4f}%.")
print(plot_dict)
```

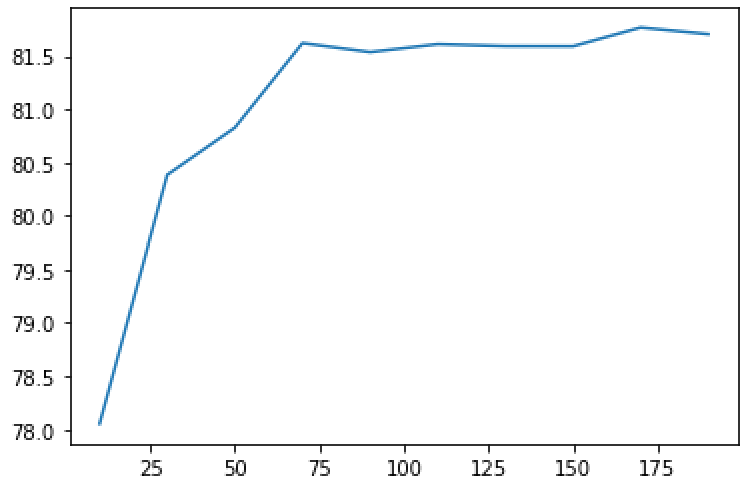
```
for n_estimator=10, auc is 78.0516%.
for n_estimator=30, auc is 80.3842%.
for n_estimator=50, auc is 80.8277%.
for n_estimator=70, auc is 81.6233%.
for n_estimator=90, auc is 81.5383%.
for n_estimator=110, auc is 81.6132%.
for n_estimator=130, auc is 81.5947%.
for n_estimator=150, auc is 81.5935%.
for n_estimator=170, auc is 81.7698%.
for n_estimator=190, auc is 81.7073%.
{10: 78.05161470406195, 30: 80.38416728358354, 50: 80.82774062792025, 70: 81.62
325999101931, 90: 81.53830750373184, 110: 81.613247733589, 130: 81.594740227429
95, 150: 81.59352662046871, 170: 81.7698030315902, 190: 81.70730227308584}
```

In [105]:
```python
#From the above the best parameter choise is given by
# ...for n_estimator=170, auc is 81.4288%.
import matplotlib.pyplot as plt

#plot: plot_dict

#plt.bar(range(len(plot_dict)), list(plot_dict.values()), align='center')
#plt.xticks(range(len(plot_dict)), list(plot_dict.keys()))
plt.plot(plot_dict.keys(), plot_dict.values())

plt.show()
```

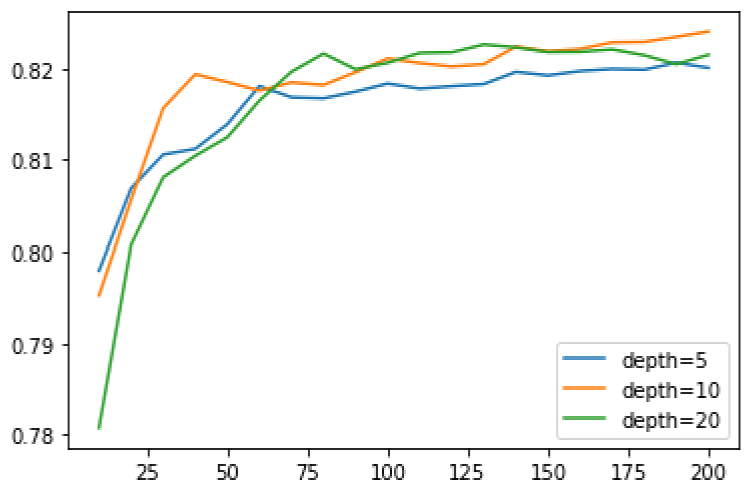


In [106]:
```
#tune the parameters for the trees in the random forest , i.e., max_depth and mir

#tune max_depth
complete_auc = {}
for depth in [5, 10, 20]:
    aucs_per_depth = []

    for i in range(10, 201, 10):
        rf = RandomForestClassifier(n_estimators=i, max_depth=depth, random_state
        rf.fit(X_Train, y_train)
        y_pred = rf.predict_proba(X_val)[:, 1]
        auc = roc_auc_score(y_val, y_pred)
        #print('%s -> %.3f' % (i, auc))
        aucs_per_depth.append(auc)
    complete_auc[depth] = aucs_per_depth

#plot auc against n_estimators per max_depth
num_trees = list(range(10, 201, 10))
plt.plot(num_trees, complete_auc[5], label='depth=5')
plt.plot(num_trees, complete_auc[10], label='depth=10')
plt.plot(num_trees, complete_auc[20], label='depth=20')
plt.legend()
plt.show()
```



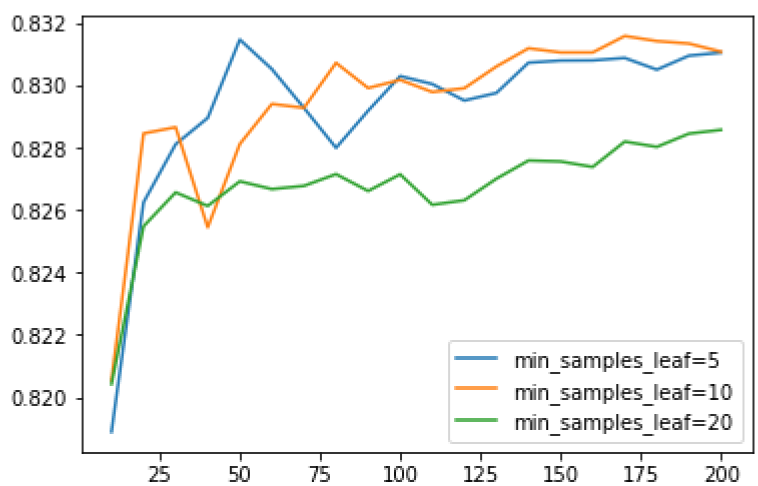
max_depth of 10 shows the best performance, so lets settle on that.

In [107]:

```
#tune min_leaf_size
complete_auc = {}
for min_leaf in [5, 10, 20]:
    aucs_per_min_leaf = []

    for i in range(10, 201, 10):
        rf = RandomForestClassifier(n_estimators=i, max_depth=10, min_samples_lea
        rf.fit(X_Train, y_train)
        y_pred = rf.predict_proba(X_val)[:, 1]
        auc = roc_auc_score(y_val, y_pred)
        #print('%s -> %.3f' % (i, auc))
        aucs_per_min_leaf.append(auc)
    complete_auc[min_leaf] = aucs_per_min_leaf

#plot auc against n_estimators per max_depth
num_trees = list(range(10, 201, 10))
plt.plot(num_trees, complete_auc[5], label='min_samples_leaf=5')
plt.plot(num_trees, complete_auc[10], label='min_samples_leaf=10')
plt.plot(num_trees, complete_auc[20], label='min_samples_leaf=20')
plt.legend()
plt.show()
```



In [108]:
```
#min_samples_leaf shows best results at the value of 5
#thus, lets train the randomforest using the tuned values

rf = RandomForestClassifier(n_estimators=170, max_depth=10, min_samples_leaf=5, r
```

Gradient Boosting Section....using Extreme Gradient Boosting (xgboost) library.

Boosting refers to sequential training of models, where each model seeks to correct the deficiencies in the nodel that operated before it. Gradient Boosting is meant to be used with trees with particular efficiency. This differs from RandomForests in that, with random forests the individual models operate in parallel and a tally of their outputs are taken to determine the final outcome of the model.

In [116]:
```
#xgboost relies on data being loaded into a DMatrix structure. This structure is
import xgboost as xgb

#create the DMatrix object for the training data
dtrain = xgb.DMatrix(X_Train, label=y_train, feature_names=dv.feature_names_)
#create the DMatrix object for the validation data
dval = xgb.DMatrix(X_val, label=y_val, feature_names=dv.feature_names_)
```

In [128]:
```
#set training parameters
xgb_params = {
    'eta': 0.3, #learning rate : the weight given to correcting the output of the
    'max_depth': 6, #tree max depth parameter
    'min_child_weight': 1, #min num. of observations in each group
    'objective': 'binary:logistic', #binary classification usage
    'eval_metric': 'auc', #the evaluation metric
    'nthread': 8, #num. of threads used in training the model, to allow parallel
    'seed': 42, #random state generator - set for reproducability
    'silent': 1
}

#set watchlist, which is a list of tuples consisting of DMatric object and a stri
#the watchlist will consist of DMatrix objects used to evaluate the model's perfo
watchlist = [(dtrain, 'train'), (dval, 'val')]
```

In [129]:
```
#train the actaul model based on the above set parameters and data

#num_boost_round indicates the number of trees,
#set it to 10 here
model = xgb.train(xgb_params, dtrain, num_boost_round=100, evals=watchlist, verbo
```

```
[14:57:07] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.3.
0/src/learner.cc:541:
Parameters: { silent } might not be used.

  This may not be accurate due to some parameters are only used in language bin
dings but
  passed down to XGBoost core.  Or some parameters are not used but slip throug
h this
  verification. Please open an issue if you find above cases.


[0]	train-auc:0.85921	val-auc:0.78128
[10]	train-auc:0.95480	val-auc:0.82279
[20]	train-auc:0.97559	val-auc:0.82299
[30]	train-auc:0.98658	val-auc:0.82289
[40]	train-auc:0.99322	val-auc:0.81959
[50]	train-auc:0.99662	val-auc:0.81963
[60]	train-auc:0.99761	val-auc:0.82005
[70]	train-auc:0.99934	val-auc:0.81690
[80]	train-auc:0.99984	val-auc:0.81633
[90]	train-auc:0.99999	val-auc:0.81766
[99]	train-auc:1.00000	val-auc:0.81554
```

The trainng set performance keeps increasing as expected with a greater number of trees. However, the performance against shows overfitting past 20 trees.

In [130]:
```
#tune parameters: eta, max_depth and min_child weight

#tune eta:
# eta: learning rate
#    if eta is too big, overfitting will occur soon, before the model matures wel
#    if eta is too small, too many trees will need to be trained before a sataisf
#    often a value of eta==0.3 is prescribed for large datasets. This dataset is 
# max_depth: same as seen above, the max depth the tree is allowed to take

xgb_params = {
    'eta': 0.1, #learning rate : reduced here to fit the smaller dataset being wo
    'max_depth': 6, #tree max depth parameter
    'min_child_weight': 1, #min num. of observations in each group
    'objective': 'binary:logistic', #binary classification usage
    'eval_metric': 'auc', #the evaluation metric
    'nthread': 8, #num. of threads used in training the model, to allow parallel 
    'seed': 42, #random state generator - set for reproducability
    'silent': 1
}
model = xgb.train(xgb_params, dtrain, num_boost_round=100, evals=watchlist, verbo
```

```
[15:01:56] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.3.
0/src/learner.cc:541:
Parameters: { silent } might not be used.

  This may not be accurate due to some parameters are only used in language bin
dings but
  passed down to XGBoost core.  Or some parameters are not used but slip throug
h this
  verification. Please open an issue if you find above cases.


[0]	train-auc:0.85921	val-auc:0.78128
[10]	train-auc:0.91751	val-auc:0.82275
[20]	train-auc:0.94190	val-auc:0.82355
[30]	train-auc:0.95361	val-auc:0.82077
[40]	train-auc:0.96239	val-auc:0.82349
[50]	train-auc:0.97009	val-auc:0.82165
[60]	train-auc:0.97598	val-auc:0.82153
[70]	train-auc:0.98125	val-auc:0.82021
[80]	train-auc:0.98456	val-auc:0.81979
[90]	train-auc:0.98705	val-auc:0.81945
[99]	train-auc:0.98891	val-auc:0.81862
```

In [131]:
```
#slight increase in performance with the modified eta of 0.1 given by: [20] trai

#tune max_depth: to 3 and 8
# min_child_weight: min num. of observations in each group

xgb_params1 = {
    'eta': 0.1, #learning rate : reduced here to fit the smaller dataset being wo
    'max_depth': 3, #tree max depth parameter
    'min_child_weight': 1, #min num. of observations in each group
    'objective': 'binary:logistic', #binary classification usage
    'eval_metric': 'auc', #the evaluation metric
    'nthread': 8, #num. of threads used in training the model, to allow parallel
    'seed': 42, #random state generator - set for reproducability
    'silent': 1
}
model1 = xgb.train(xgb_params1, dtrain, num_boost_round=100, evals=watchlist, ver

xgb_params2 = {
    'eta': 0.1, #learning rate : reduced here to fit the smaller dataset being wo
    'max_depth': 8, #tree max depth parameter
    'min_child_weight': 1, #min num. of observations in each group
    'objective': 'binary:logistic', #binary classification usage
    'eval_metric': 'auc', #the evaluation metric
    'nthread': 8, #num. of threads used in training the model, to allow parallel
    'seed': 42, #random state generator - set for reproducability
    'silent': 1
}
model2 = xgb.train(xgb_params2, dtrain, num_boost_round=100, evals=watchlist, ver
```

```
[15:05:59] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.3.
0/src/learner.cc:541:
Parameters: { silent } might not be used.

  This may not be accurate due to some parameters are only used in language bin
dings but
  passed down to XGBoost core.  Or some parameters are not used but slip throug
h this
  verification. Please open an issue if you find above cases.


[0]	train-auc:0.77023	val-auc:0.74305
[10]	train-auc:0.83535	val-auc:0.80944
[20]	train-auc:0.86023	val-auc:0.82033
[30]	train-auc:0.87601	val-auc:0.82754
[40]	train-auc:0.88668	val-auc:0.82910
[50]	train-auc:0.89335	val-auc:0.83202
[60]	train-auc:0.89893	val-auc:0.83166
[70]	train-auc:0.90400	val-auc:0.83382
[80]	train-auc:0.90700	val-auc:0.83534
[90]	train-auc:0.90992	val-auc:0.83551
[99]	train-auc:0.91260	val-auc:0.83480
[15:05:59] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.3.
0/src/learner.cc:541:
Parameters: { silent } might not be used.

  This may not be accurate due to some parameters are only used in language bin
```

```
dings but
  passed down to XGBoost core.  Or some parameters are not used but slip throug
h this
  verification. Please open an issue if you find above cases.


[0]	train-auc:0.89823	val-auc:0.77731
[10]	train-auc:0.96442	val-auc:0.81075
[20]	train-auc:0.98214	val-auc:0.81542
[30]	train-auc:0.98925	val-auc:0.81375
[40]	train-auc:0.99304	val-auc:0.81527
[50]	train-auc:0.99641	val-auc:0.81479
[60]	train-auc:0.99760	val-auc:0.81515
[70]	train-auc:0.99849	val-auc:0.81566
[80]	train-auc:0.99880	val-auc:0.81725
[90]	train-auc:0.99909	val-auc:0.81775
[99]	train-auc:0.99940	val-auc:0.81690
```

The best performance given by eta:0.1, max_depth:3 --> [90] train-auc:0.90992 val-auc:0.83551
this likely shows benefits to limiting the tree-depth in reducing overfitting

In [133]:
```
#tune min_child_weight: to 1,10 and 30
# min_child_weight: min num. of observations in each group

xgb_params1 = {
    'eta': 0.1, #learning rate : reduced here to fit the smaller dataset being wo
    'max_depth': 3, #tree max depth parameter
    'min_child_weight': 1, #min num. of observations in each group
    'objective': 'binary:logistic', #binary classification usage
    'eval_metric': 'auc', #the evaluation metric
    'nthread': 8, #num. of threads used in training the model, to allow parallel
    'seed': 42, #random state generator - set for reproducability
    'silent': 1
}
model1 = xgb.train(xgb_params1, dtrain, num_boost_round=100, evals=watchlist, ver

xgb_params2 = {
    'eta': 0.1, #learning rate : reduced here to fit the smaller dataset being wo
    'max_depth': 3, #tree max depth parameter
    'min_child_weight': 10, #min num. of observations in each group
    'objective': 'binary:logistic', #binary classification usage
    'eval_metric': 'auc', #the evaluation metric
    'nthread': 8, #num. of threads used in training the model, to allow parallel
    'seed': 42, #random state generator - set for reproducability
    'silent': 1
}
model3 = xgb.train(xgb_params2, dtrain, num_boost_round=100, evals=watchlist, ver


xgb_params3 = {
    'eta': 0.1, #learning rate : reduced here to fit the smaller dataset being wo
    'max_depth': 3, #tree max depth parameter
    'min_child_weight': 30, #min num. of observations in each group
    'objective': 'binary:logistic', #binary classification usage
    'eval_metric': 'auc', #the evaluation metric
    'nthread': 8, #num. of threads used in training the model, to allow parallel
    'seed': 42, #random state generator - set for reproducability
    'silent': 1
}
model3 = xgb.train(xgb_params3, dtrain, num_boost_round=100, evals=watchlist, ver
```

```
[15:16:19] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.3.
0/src/learner.cc:541:
Parameters: { silent } might not be used.

  This may not be accurate due to some parameters are only used in language bin
dings but
  passed down to XGBoost core.  Or some parameters are not used but slip throug
h this
  verification. Please open an issue if you find above cases.


[0]     train-auc:0.77023       val-auc:0.74305
[10]    train-auc:0.83535       val-auc:0.80944
[20]    train-auc:0.86023       val-auc:0.82033
```

```
[30]	train-auc:0.87601	val-auc:0.82754
[40]	train-auc:0.88668	val-auc:0.82910
[50]	train-auc:0.89335	val-auc:0.83202
[60]	train-auc:0.89893	val-auc:0.83166
[70]	train-auc:0.90400	val-auc:0.83382
[80]	train-auc:0.90700	val-auc:0.83534
[90]	train-auc:0.90992	val-auc:0.83551
[99]	train-auc:0.91260	val-auc:0.83480
[15:16:19] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.3.
0/src/learner.cc:541:
Parameters: { silent } might not be used.

  This may not be accurate due to some parameters are only used in language bin
dings but
  passed down to XGBoost core.  Or some parameters are not used but slip throug
h this
  verification. Please open an issue if you find above cases.


[0]	train-auc:0.76953	val-auc:0.74606
[10]	train-auc:0.83855	val-auc:0.80951
[20]	train-auc:0.85991	val-auc:0.82070
[30]	train-auc:0.87058	val-auc:0.82846
[40]	train-auc:0.87910	val-auc:0.83130
[50]	train-auc:0.88577	val-auc:0.83304
[60]	train-auc:0.89237	val-auc:0.83493
[70]	train-auc:0.89619	val-auc:0.83597
[80]	train-auc:0.89956	val-auc:0.83677
[90]	train-auc:0.90196	val-auc:0.83661
[99]	train-auc:0.90421	val-auc:0.83691
[15:16:20] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.3.
0/src/learner.cc:541:
Parameters: { silent } might not be used.

  This may not be accurate due to some parameters are only used in language bin
dings but
  passed down to XGBoost core.  Or some parameters are not used but slip throug
h this
  verification. Please open an issue if you find above cases.


[0]	train-auc:0.76905	val-auc:0.74605
[10]	train-auc:0.83204	val-auc:0.80962
[20]	train-auc:0.85303	val-auc:0.81699
[30]	train-auc:0.86301	val-auc:0.82516
[40]	train-auc:0.87069	val-auc:0.82802
[50]	train-auc:0.87491	val-auc:0.82938
[60]	train-auc:0.87915	val-auc:0.83211
[70]	train-auc:0.88186	val-auc:0.83307
[80]	train-auc:0.88404	val-auc:0.83459
[90]	train-auc:0.88648	val-auc:0.83528
[99]	train-auc:0.88819	val-auc:0.83604
```

In [140]:

```
#min_child_weight shows the best results

xgb_params1 = {
    'eta': 0.1, #learning rate : reduced here to fit the smaller dataset being wo
    'max_depth': 3, #tree max depth parameter
    'min_child_weight': 10, #min num. of observations in each group
    'objective': 'binary:logistic', #binary classification usage
    'eval_metric': 'auc', #the evaluation metric
    'nthread': 8, #num. of threads used in training the model, to allow parallel
    'seed': 42, #random state generator - set for reproducability
    'silent': 1
}
model1 = xgb.train(xgb_params1, dtrain, num_boost_round=100, evals=watchlist, ver

xgb_params2 = {
    'eta': 0.1, #learning rate : reduced here to fit the smaller dataset being wo
    'max_depth': 3, #tree max depth parameter
    'min_child_weight': 10, #min num. of observations in each group
    'objective': 'binary:logistic', #binary classification usage
    'eval_metric': 'auc', #the evaluation metric
    'nthread': 8, #num. of threads used in training the model, to allow parallel
    'seed': 42, #random state generator - set for reproducability
    'silent': 1
}
model2 = xgb.train(xgb_params2, dtrain, num_boost_round=99, evals=watchlist, verb

xgb_params3 = {
    'eta': 0.1, #learning rate : reduced here to fit the smaller dataset being wo
    'max_depth': 3, #tree max depth parameter
    'min_child_weight': 10, #min num. of observations in each group
    'objective': 'binary:logistic', #binary classification usage
    'eval_metric': 'auc', #the evaluation metric
    'nthread': 8, #num. of threads used in training the model, to allow parallel
    'seed': 42, #random state generator - set for reproducability
    'silent': 1
}
model3 = xgb.train(xgb_params3, dtrain, num_boost_round=150, evals=watchlist, ver
```

```
[15:21:02] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.3.
0/src/learner.cc:541:
Parameters: { silent } might not be used.

  This may not be accurate due to some parameters are only used in language bin
dings but
  passed down to XGBoost core.  Or some parameters are not used but slip throug
h this
  verification. Please open an issue if you find above cases.


[0]     train-auc:0.76953       val-auc:0.74606
[10]    train-auc:0.83855       val-auc:0.80951
[20]    train-auc:0.85991       val-auc:0.82070
[30]    train-auc:0.87058       val-auc:0.82846
[40]    train-auc:0.87910       val-auc:0.83130
[50]    train-auc:0.88577       val-auc:0.83304
[60]    train-auc:0.89237       val-auc:0.83493
```

```
[70]	train-auc:0.89619	val-auc:0.83597
[80]	train-auc:0.89956	val-auc:0.83677
[90]	train-auc:0.90196	val-auc:0.83661
[99]	train-auc:0.90421	val-auc:0.83691
[15:21:02] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.3.
0/src/learner.cc:541:
Parameters: { silent } might not be used.

  This may not be accurate due to some parameters are only used in language bin
dings but
  passed down to XGBoost core.  Or some parameters are not used but slip throug
h this
  verification. Please open an issue if you find above cases.


[0]	train-auc:0.76953	val-auc:0.74606
[10]	train-auc:0.83855	val-auc:0.80951
[20]	train-auc:0.85991	val-auc:0.82070
[30]	train-auc:0.87058	val-auc:0.82846
[40]	train-auc:0.87910	val-auc:0.83130
[50]	train-auc:0.88577	val-auc:0.83304
[60]	train-auc:0.89237	val-auc:0.83493
[70]	train-auc:0.89619	val-auc:0.83597
[80]	train-auc:0.89956	val-auc:0.83677
[90]	train-auc:0.90196	val-auc:0.83661
[98]	train-auc:0.90405	val-auc:0.83685
[15:21:03] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.3.
0/src/learner.cc:541:
Parameters: { silent } might not be used.

  This may not be accurate due to some parameters are only used in language bin
dings but
  passed down to XGBoost core.  Or some parameters are not used but slip throug
h this
  verification. Please open an issue if you find above cases.


[0]	train-auc:0.76953	val-auc:0.74606
[10]	train-auc:0.83855	val-auc:0.80951
[20]	train-auc:0.85991	val-auc:0.82070
[30]	train-auc:0.87058	val-auc:0.82846
[40]	train-auc:0.87910	val-auc:0.83130
[50]	train-auc:0.88577	val-auc:0.83304
[60]	train-auc:0.89237	val-auc:0.83493
[70]	train-auc:0.89619	val-auc:0.83597
[80]	train-auc:0.89956	val-auc:0.83677
[90]	train-auc:0.90196	val-auc:0.83661
[100]	train-auc:0.90504	val-auc:0.83671
[110]	train-auc:0.90754	val-auc:0.83747
[120]	train-auc:0.90968	val-auc:0.83642
[130]	train-auc:0.91159	val-auc:0.83654
[140]	train-auc:0.91365	val-auc:0.83556
[149]	train-auc:0.91556	val-auc:0.83523
```

In [141]:

```
#taking num. trees to be 110 as seen above for best performance of 83.747%
#the final model is below:

xgb_params_final = {
    'eta': 0.1, #learning rate : reduced here to fit the smaller dataset being wo
    'max_depth': 3, #tree max depth parameter
    'min_child_weight': 10, #min num. of observations in each group
    'objective': 'binary:logistic', #binary classification usage
    'eval_metric': 'auc', #the evaluation metric
    'nthread': 8, #num. of threads used in training the model, to allow parallel
    'seed': 42, #random state generator - set for reproducability
    'silent': 1
}
num_trees = 110
model_final = xgb.train(xgb_params_final, dtrain, num_boost_round=num_trees)
```

```
[15:24:35] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.3.
0/src/learner.cc:541:
Parameters: { silent } might not be used.

  This may not be accurate due to some parameters are only used in language bin
dings but
  passed down to XGBoost core.  Or some parameters are not used but slip throug
h this
  verification. Please open an issue if you find above cases.
```

In [ ]:

```
```